Deep Learning for NLP
Part 2

CS224N

Christopher Manning

(Many slides borrowed from ACL 2012/NAACL 2013
Tutorials by me, Richard Socher and Yoshua Bengio)

Advantages of Deep Learning

#1 Learning representations

Handcrafting features is time-consuming

The features are often both over-specified and incomplete

The work has to be done again for each task/domain/…

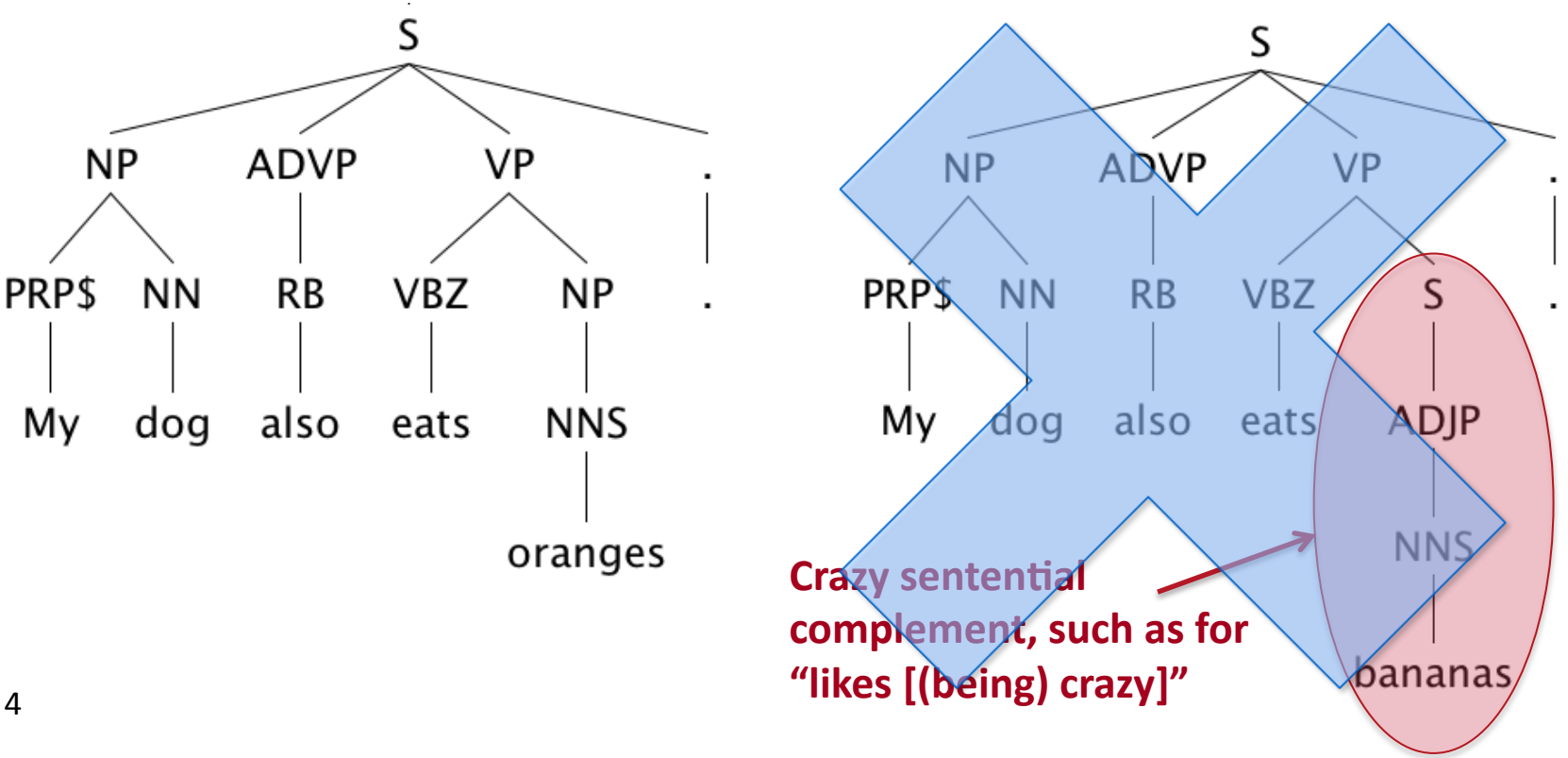
Humans develop representations for learning and reasoning

Our computers should do the same

Deep learning provides a way of doing this
Machine Learning

#2 The need for distributed representations

Current NLP systems are incredibly fragile because of their atomic symbol representations



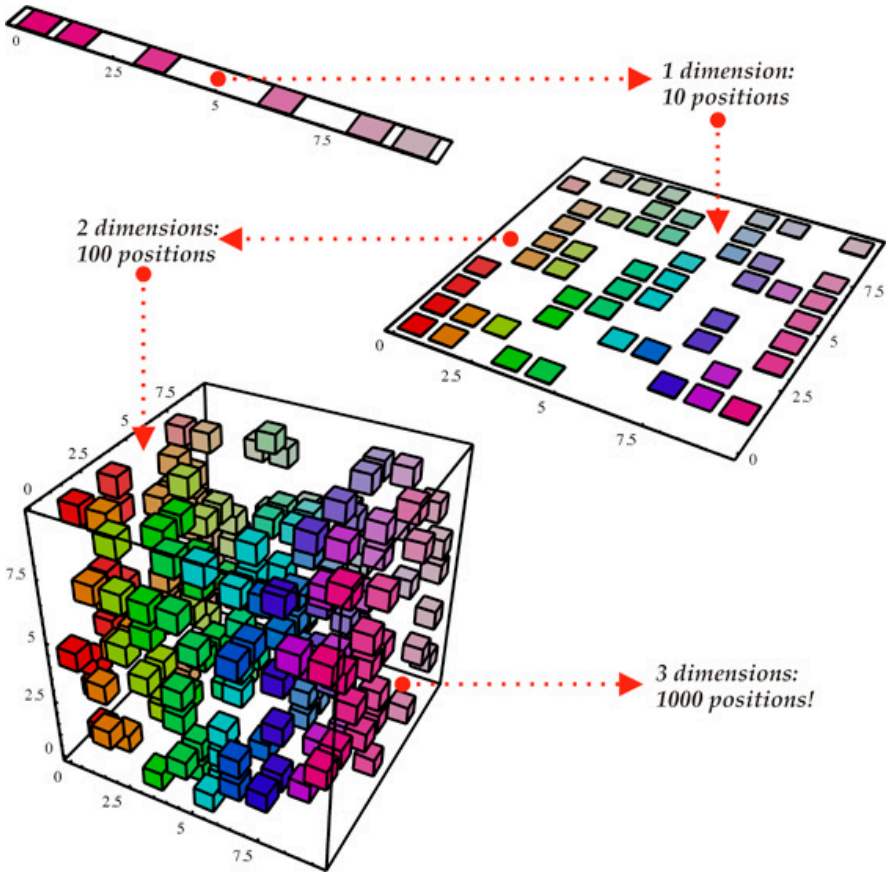
Distributed representations deal with the curse of dimensionality

Generalizing locally (e.g., nearest neighbors) requires representative examples for all relevant variations!

Classic solutions:

- Manual feature design
- Assuming a smooth target function (e.g., linear models)
- Kernel methods (linear in terms of kernel based on data points)

Neural networks parameterize and learn a "similarity" kernel



#3 Unsupervised feature learning

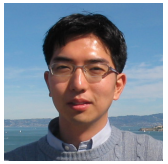
Today, most practical, good NLP& ML methods require labeled training data (i.e., supervised learning)

- But almost all data is unlabeled

Most information must be acquired **unsupervised**

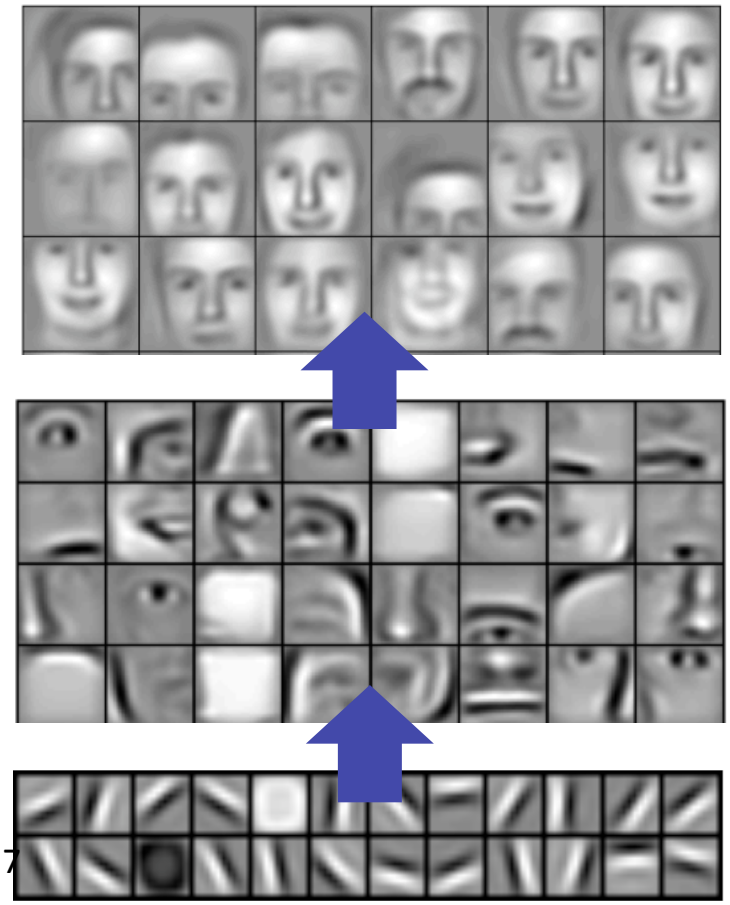
- Fortunately, a good model of observed data can really help you learn classification decisions

#4 Learning multiple levels of representation



[Lee et al. ICML 2009; Lee et al. NIPS 2009]

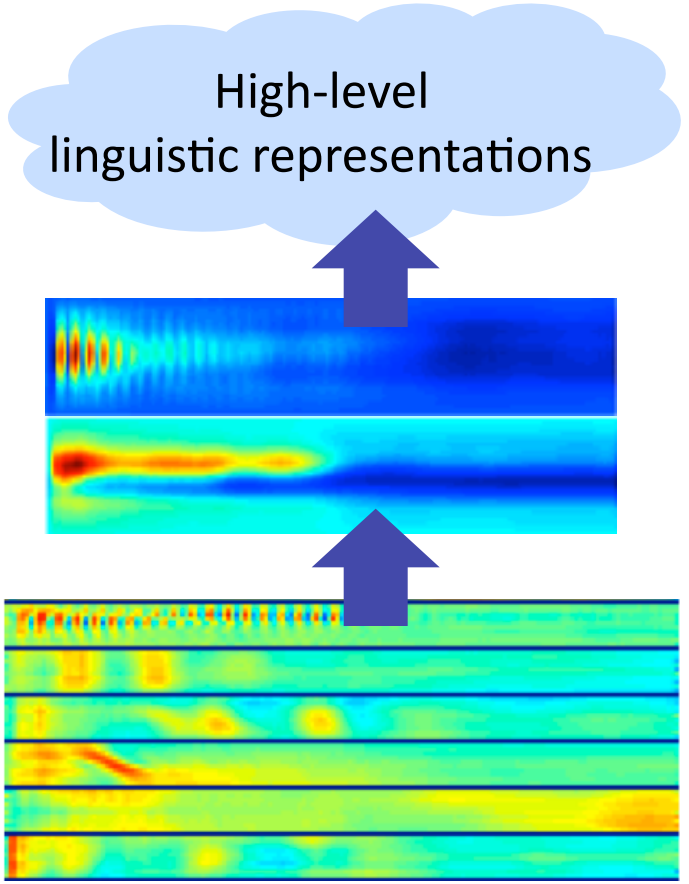
Successive model layers learn deeper intermediate representations



Layer 3

Layer 2

Layer 1

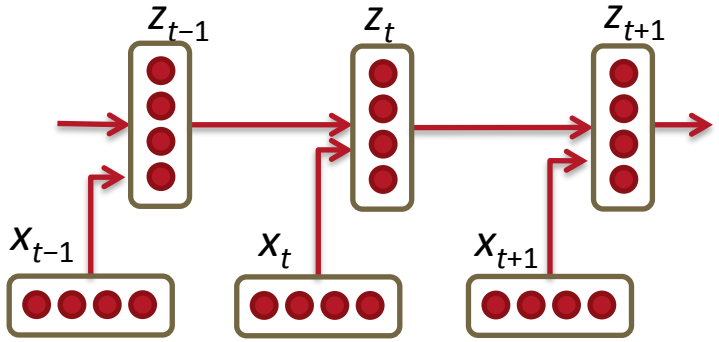


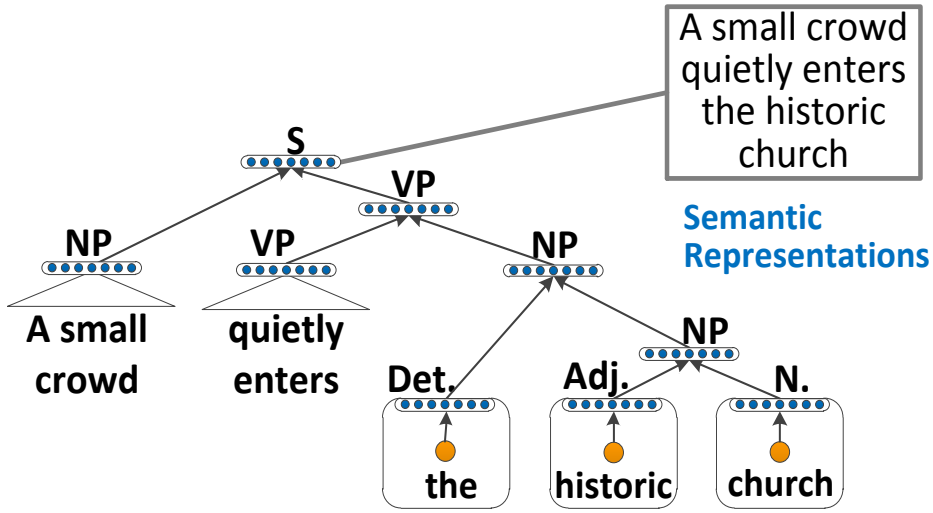
#4 Handling the recursivity of Language

Human sentences are composed from words and phrases

We need compositionality in our ML models

Recursion: the same operator (same parameters) is applied repeatedly on different components





#5 Why now?

Despite prior investigation and understanding of many of the algorithmic techniques …

Before 2006 training deep architectures was **unsuccessful** ☹

What has changed?

- New methods for unsupervised pre-training have been developed (Restricted Boltzmann Machines = RBMs, autoencoders, contrastive estimation, etc.)
- More efficient parameter estimation methods
- Better understanding of model regularization
- **More data** and **more computational power**

Deep Learning models have already achieved impressive results for HLT

Neural Language Model

[Mikolov et al. Interspeech 2011]

Model \ WSJ ASR task	Eval WER
KN5 Baseline	**17.2**
Discriminative LM	**16.9**
Recurrent NN combination	**14.4**

MSR MAVIS Speech System

[Dahl et al. 2012; Seide et al. 2011; following Mohamed et al. 2011]

"The algorithms represent the first time a company has released a deep-neural-networks (DNN)-based speech-recognition algorithm in a commercial product."

Acoustic model & training	Recog \ WER	RT03S FSH	Hub5 SWB
GMM 40-mix, BMMI, SWB 309h	1-pass –adapt	**27.4**	**23.6**
DBN-DNN 7 layer x 2048, SWB 309h	1-pass –adapt	**18.5** (–33%)	**16.1** (–32%)
GMM 72-mix, BMMI, FSH 2000h	*k*-pass +adapt	**18.6**	**17.1**

Deep Learn Models Have Interesting Performance Characteristics

Deep learning models can now be very fast in some circumstances

- SENNA [Collobert et al. 2011] can do POS or NER faster than other SOTA taggers (16x to 122x), using 25x less memory
- WSJ POS 97.29% acc; CoNLL NER 89.59% F1; CoNLL Chunking 94.32% F1

Changes in computing technology favor deep learning

- In NLP, speed has traditionally come from exploiting sparsity
- But with modern machines, branches and widely spaced memory accesses are costly
- Uniform parallel operations on dense vectors are faster

These trends are even stronger with multi-core CPUs and GPUs

Deep Learning
General Strategy and Tricks

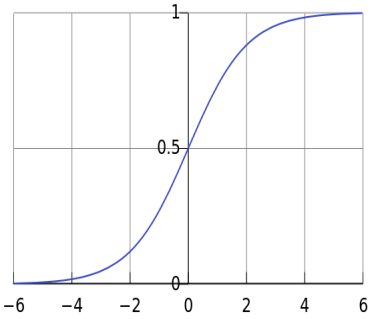
General Strategy

1. Select network structure appropriate for problem
   1. Structure: Single words, fixed windows vs. Recursive Sentence Based vs. Bag of words
   2. Nonlinearity
2. Check for implementation bugs with gradient checks
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
   1. If not, change model structure or make model "larger"
   2. If you can overfit: Regularize

Non-Linearities: What's used

logistic ("sigmoid")

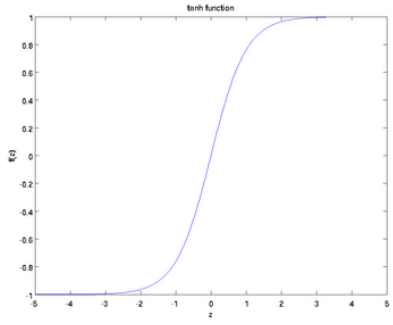
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



$$f'(z) = 1 - f(z)^2$$

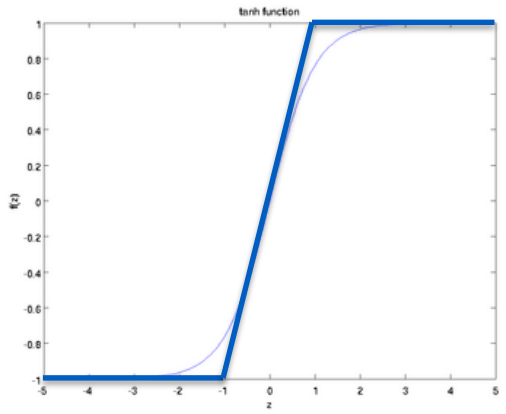
tanh is just a rescaled and shifted sigmoid $\tanh(z) = 2\text{logistic}(2z) - 1$

tanh is often used and often performs better for deep nets

Non-linearities: There are various other choices

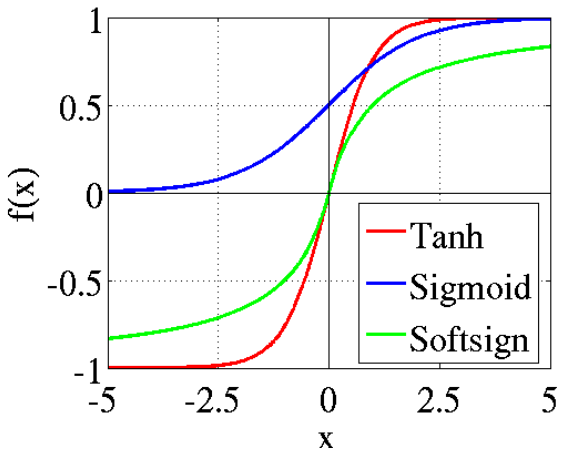
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



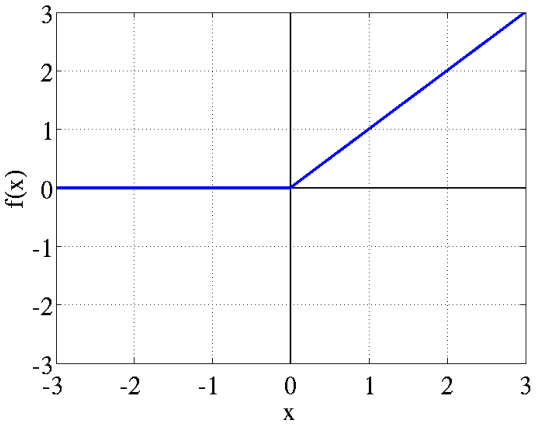
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



linear rectifier (ReLU)

$$\text{rect}(z) = \max(z, 0)$$



- hard tanh similar but computationally cheaper than tanh and saturates hard.
- [Glorot and Bengio *AISTATS* 2010, 2011] discuss softsign and rectifier
  - **Rectified linear unit** is becoming standard choice

Gradient Checks are Awesome!

- Allows you to know that there are no bugs in your neural network implementation! (But makes it run really slow.)
- Steps:
  1. Implement your gradient
  2. Implement a finite difference computation by looping through the parameters of your network, adding and subtracting a small epsilon (~10^{-4}) and estimate derivatives

$$g_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2 \times \text{EPSILON}}. \qquad \theta^{(i+)} = \theta + \text{EPSILON} \times \vec{e}_i$$

  3. Compare the two and make sure they are almost the same

General Strategy

1. Select appropriate Network Structure
   1. Structure: Single words, fixed windows vs Recursive Sentence Based vs Bag of words
   2. Nonlinearity
2. Check for implementation bugs with gradient check
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
   1. If not, change model structure or make model "larger"
   2. If you can overfit: Regularize

Parameter Initialization

- Parameter Initialization can be very important for success!
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target).
- Initialize weights ~ Uniform(−*r*, *r*), *r* inversely proportional to fan-in (previous layer size) and fan-out (next layer size):

$$\sqrt{6/(\text{fan-in} + \text{fan-out})}$$

for tanh units, and 4x bigger for sigmoid units [Glorot AISTATS 2010]

- Pre-training with Restricted Boltzmann machines

Stochastic Gradient Descent (SGD)

- Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

- L = loss function, z_t = current example, θ = parameter vector, and ε_t = learning rate.
- Ordinary gradient descent as a batch method is very slow, should never be used. Use 2nd order batch method such as L-BFGS.
- On large datasets, SGD usually wins over all batch methods. On smaller datasets L-BFGS or Conjugate Gradients win. Large-batch L-BFGS extends the reach of L-BFGS [Le et al. ICML 2011].
- Mini-batch SGD can make implementations much faster

Learning Rates

- Simplest recipe: keep it fixed and use the same for all parameters.
- Collobert scales them by the inverse of square root of the fan-in of each neuron
- Better results can generally be obtained by allowing learning rates to decrease, typically in $O(1/t)$ because of theoretical convergence guarantees, e.g., $\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$ with hyper-parameters ϵ_0 and τ
- Better yet: No hand-set learning rates by using methods like AdaGrad (Duchi, Hazan, & Singer 2011) [but may converge too soon – try resetting accumulated gradients]

General Strategy

1. Select appropriate Network Structure
    1. Structure: Single words, fixed windows vs Recursive Sentence Based vs Bag of words
    2. Nonlinearity
2. Check for implementation bugs with gradient check
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
    1. If not, change model structure or make model "larger"
    2. If you can overfit: Regularize

Assuming you found the right network structure, implemented it correctly, optimized it properly, you can make your model totally overfit on your training data (99%+ accuracy)

- If not, change architecture, make model bigger, fix optimization
- If yes, now, it's time to regularize the network

Prevent Overfitting: Model Size and Regularization

- Simple first step: Reduce model size by lowering number of units and layers and other parameters
- Standard L1 or L2 regularization on weights
- Early Stopping: Use parameters that gave best validation error
- Sparsity constraints on hidden activations, e.g., add to cost:

$$KL\left(1/N \sum_{n=1}^{N} a_i^{(n)} \| 0.0001\right)$$

Prevent Feature Co-adaptation

Dropout (Hinton et al. 2012) http://jmlr.org/papers/v15/srivastava14a.html

- Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
- Test time: halve the model weights (now twice as many)
- This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
- A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where weights are set in the context of all others)
- Can be thought of as a form of model bagging
- It acts as a strong regularizer; see (Wager et al. 2013) http://arxiv.org/abs/1307.1493

Deep Learning Tricks of the Trade

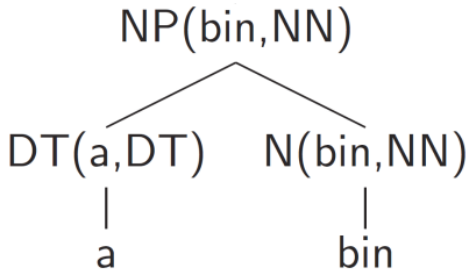
- Y. Bengio (2012), "Practical Recommendations for Gradient-Based Training of Deep Architectures" http://arxiv.org/abs/1206.5533
  - Unsupervised pre-training
  - Stochastic gradient descent and setting learning rates
  - Main hyper-parameters
    - Learning rate schedule & early stopping
    - Minibatches
    - Parameter initialization
    - Number of hidden units
    - L1 or L2 weight decay
    - Sparsity regularization
  - Debugging → use finite difference gradient checks
  - How to efficiently search for hyper-parameter configurations

TREE STRUCTURES WITH CONTINUOUS VECTORS

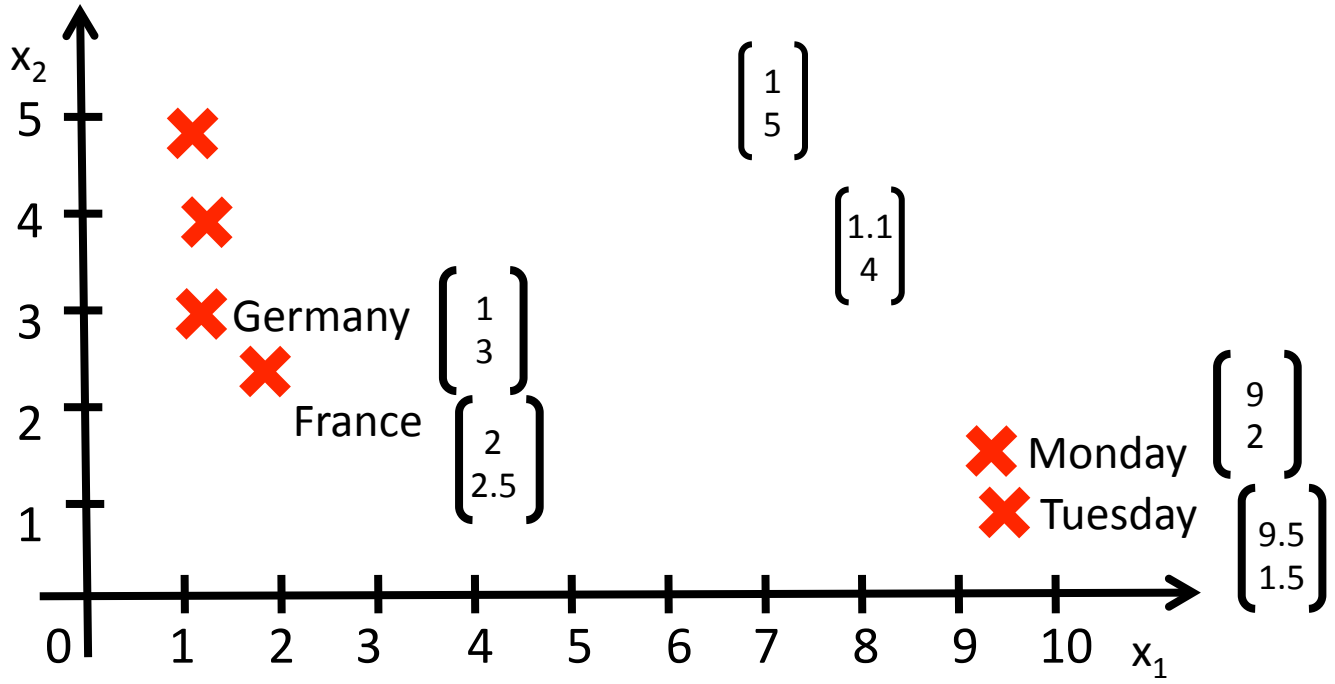
Syntactic Phrase Representations in Parsing

- Usually coarse discrete categories such as NP, PP
- Better results with more fine-grained categories
  - capture phrases with similar behavior

- Lexicalization (Collins, 2003)
  - Sparsity problems require backoff strategies
  - Discrete word indices do not capture the continuous notion of word similarity



Representing Phrases as Vectors



Vector for single words are useful as features but limited

the country of my birth

the place where I was born

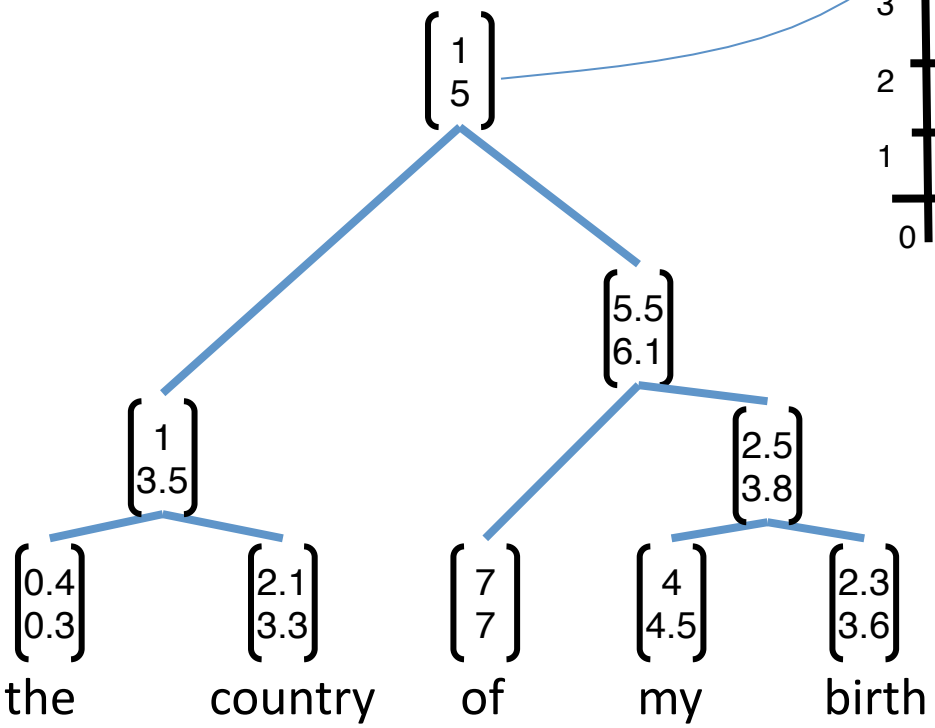
Can we extend ideas of word vector spaces to phrases?

If the vector space captures syntactic and semantic information, the vectors can be used as features for parsing and interpretation

How should we map phrases into a vector space?

Use the principle of compositionality!

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.



Model jointly learns compositional vector representations and tree structure.

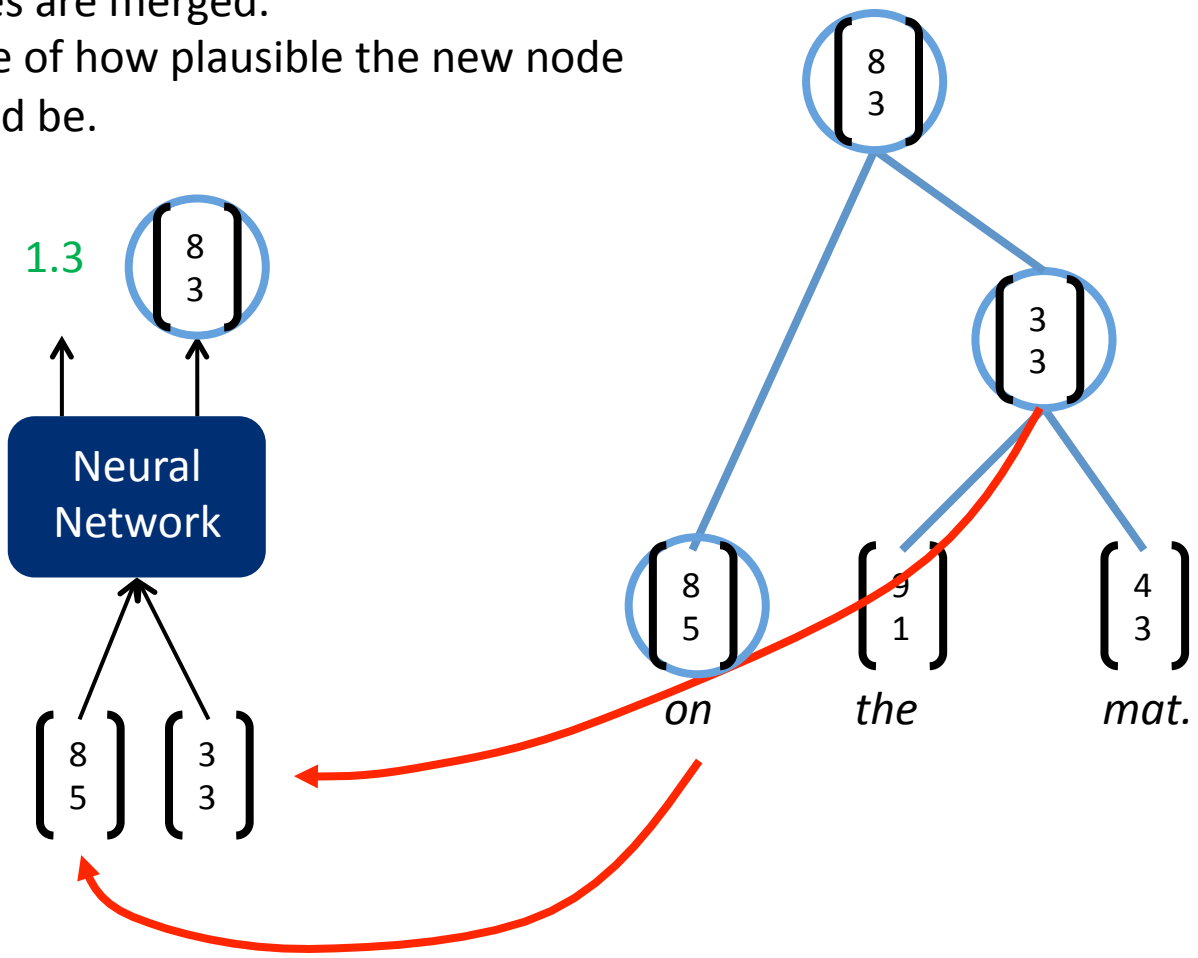
Recursive Neural Networks for Phrase Vectors

Basic computational unit: Recursive Neural Network

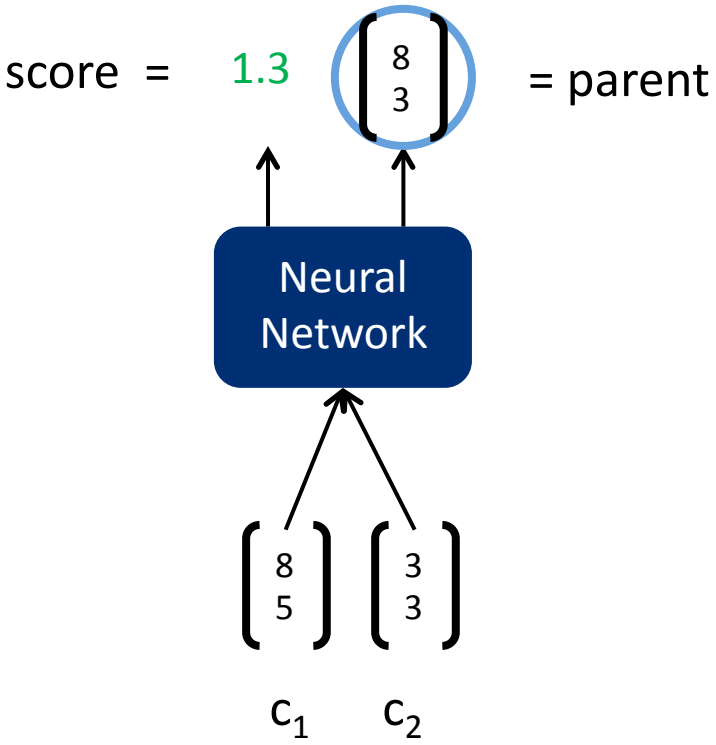
Inputs: two candidate children's representations

Outputs:

1. The semantic representation if the two nodes are merged.
2. Score of how plausible the new node would be.



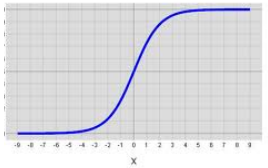
Recursive Neural Network Definition



$$\text{score} = V^T p$$

$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right),$$

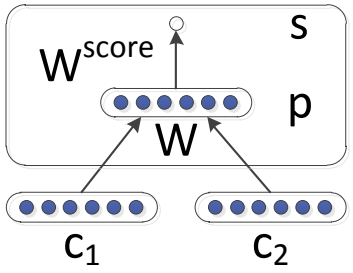
where tanh:



- Socher, Manning, and Ng (ICML, 2011)
- Related Recursive Neural Network based approaches
  - Previous RNN work (Goller & Küchler (1996), Costa et al. (2003)) assumed fixed tree structure or used binary vectors
  - Henderson (2003), Titov & Henderson (2007) use NN for parse decision prediction but not representation
  - Pollack (1990): Recursive auto-associative memories (RAAMs)

Initial Recursive Neural Networks

Only a single weight matrix RNN



Could capture some phenomena

Not adequate for more complex, higher order composition and parsing long sentences

The composition function is the same for all syntactic categories, punctuation, etc.

Slow because every potential score requires matrix-vector product

Discussion: Simple RNN

State of the art in paraphrase detection (Socher et al. 2011a), sentiment analysis (Socher et al., 2011b)

Semantic similarity / nearest neighbors

Knight-Ridder would n't comment on the offer

1. Harsco declined to say what country placed the order
2. Coastal would n't disclose the terms

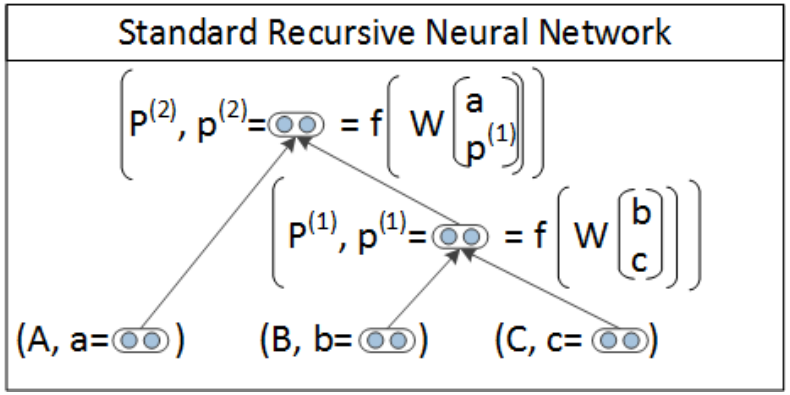
Paraphrase detection

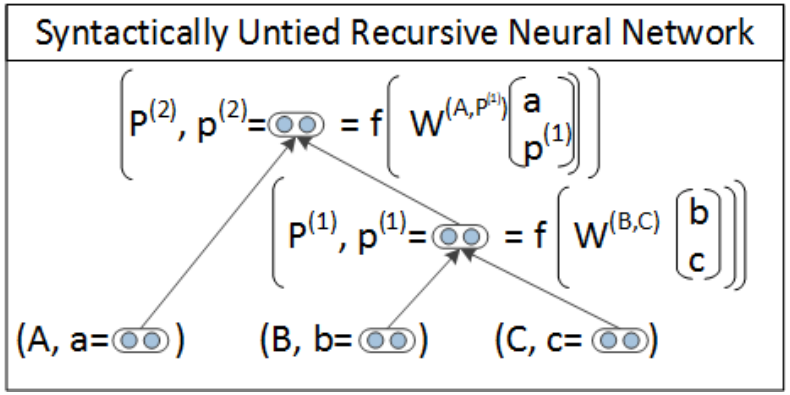
System	P	R	F1
Bannard & Callison-Burch 2005	0.28	0.12	0.17
Callison-Burch 2008 CB7	0.52	0.17	0.26
RNN (this work)	0.38	**0.64**	**0.48**
RNN-CB7 (RNN & CB7 combined)	**0.53**	0.20	0.29

Solution 2: PCFG + Syntactically-Untied RNN

Hypotheses:

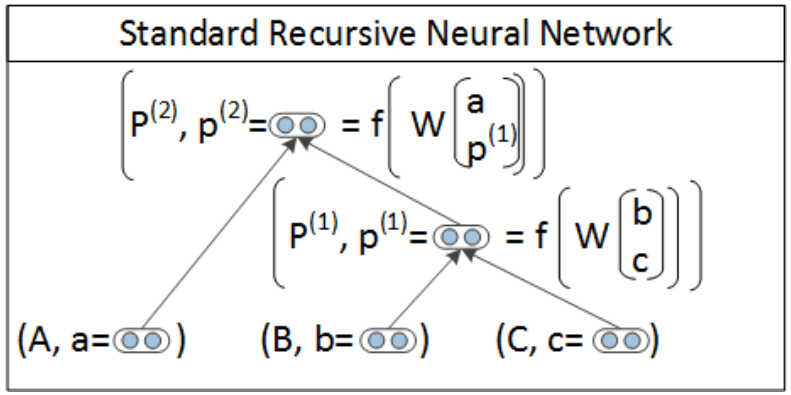
- A symbolic CFG backbone is quite adequate for basic syntactic structure
- An RNN can do a fairly good job for meaning composition
- It would do better if we allow a different composition matrix for different syntactic environments

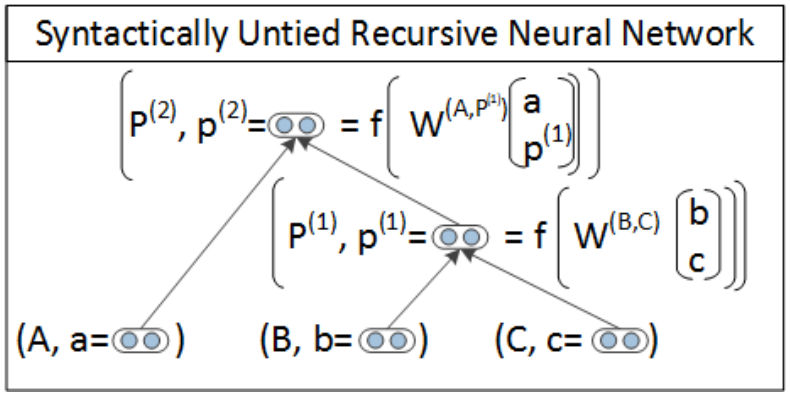




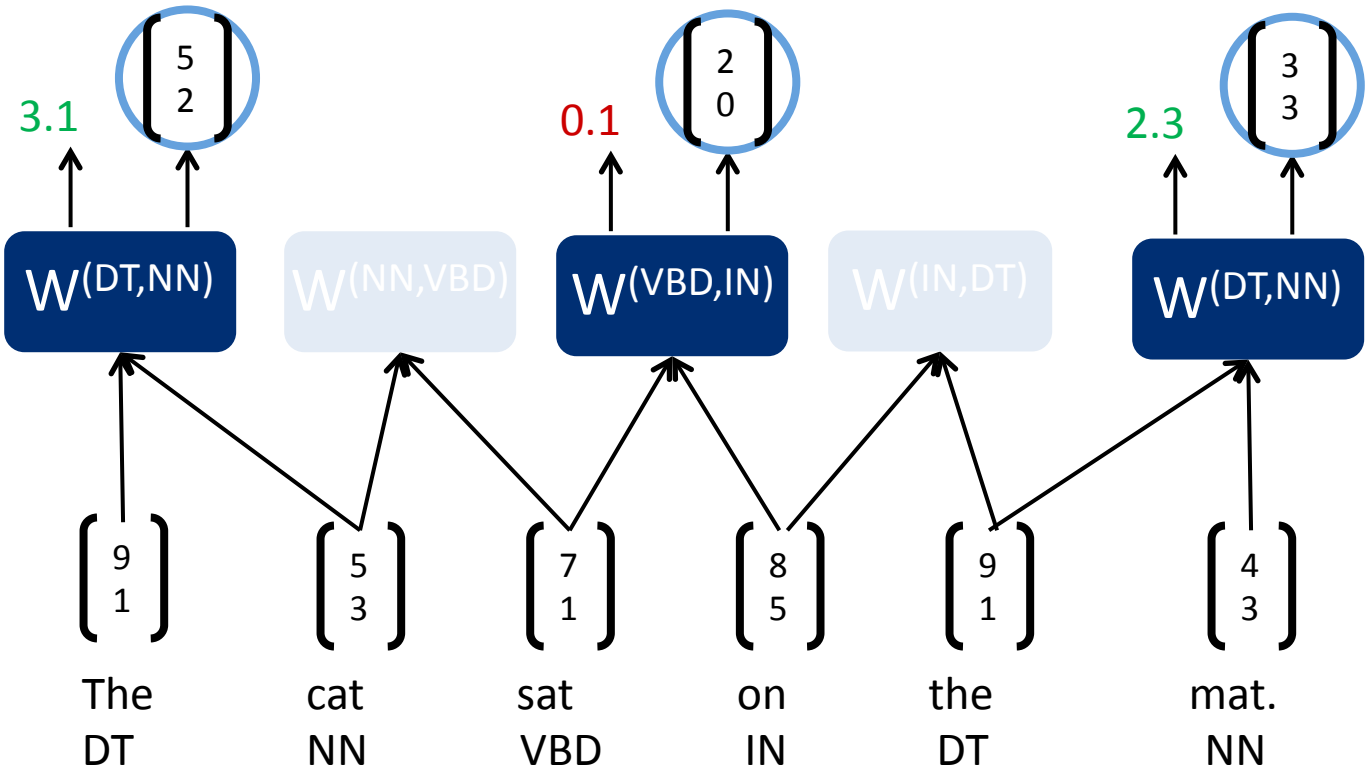
Solution 2: PCFG + Syntactically-Untied RNN

- We use the discrete syntactic categories of the children to choose the continuous composition function
- Compute score using a linear combination of the RNN score + log likelihood from the PCFG rule
- PCFG backbone gives us speed by letting us prune unlikely candidates

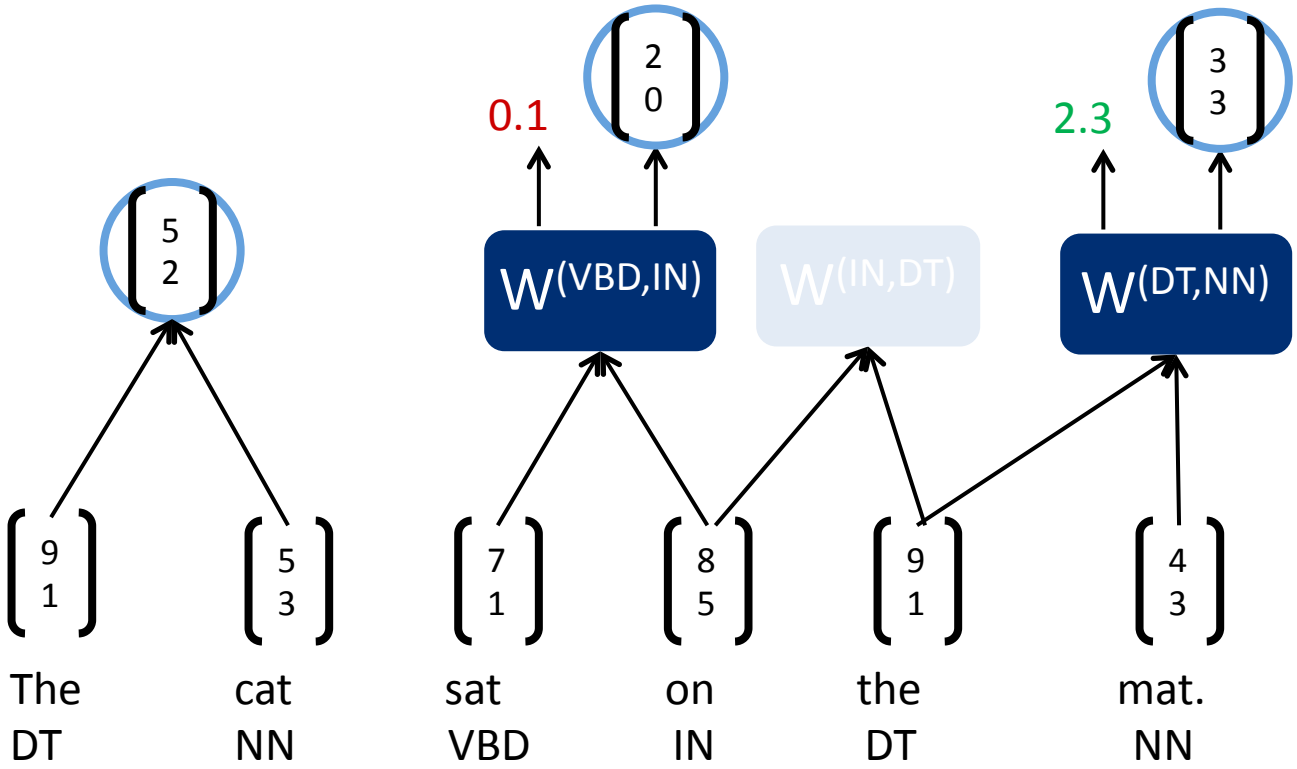




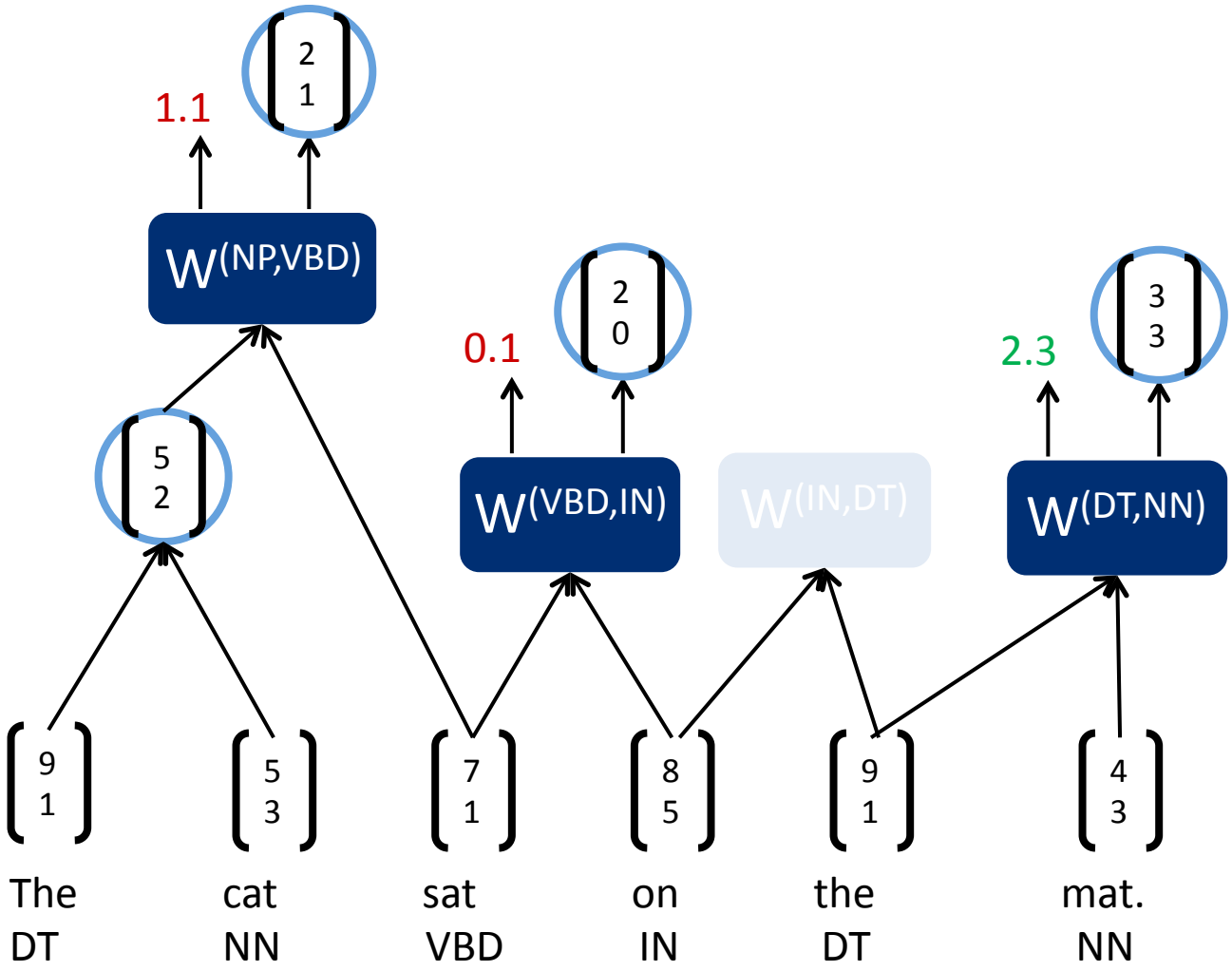
SU-RNN Procedure on Candidate Sentence



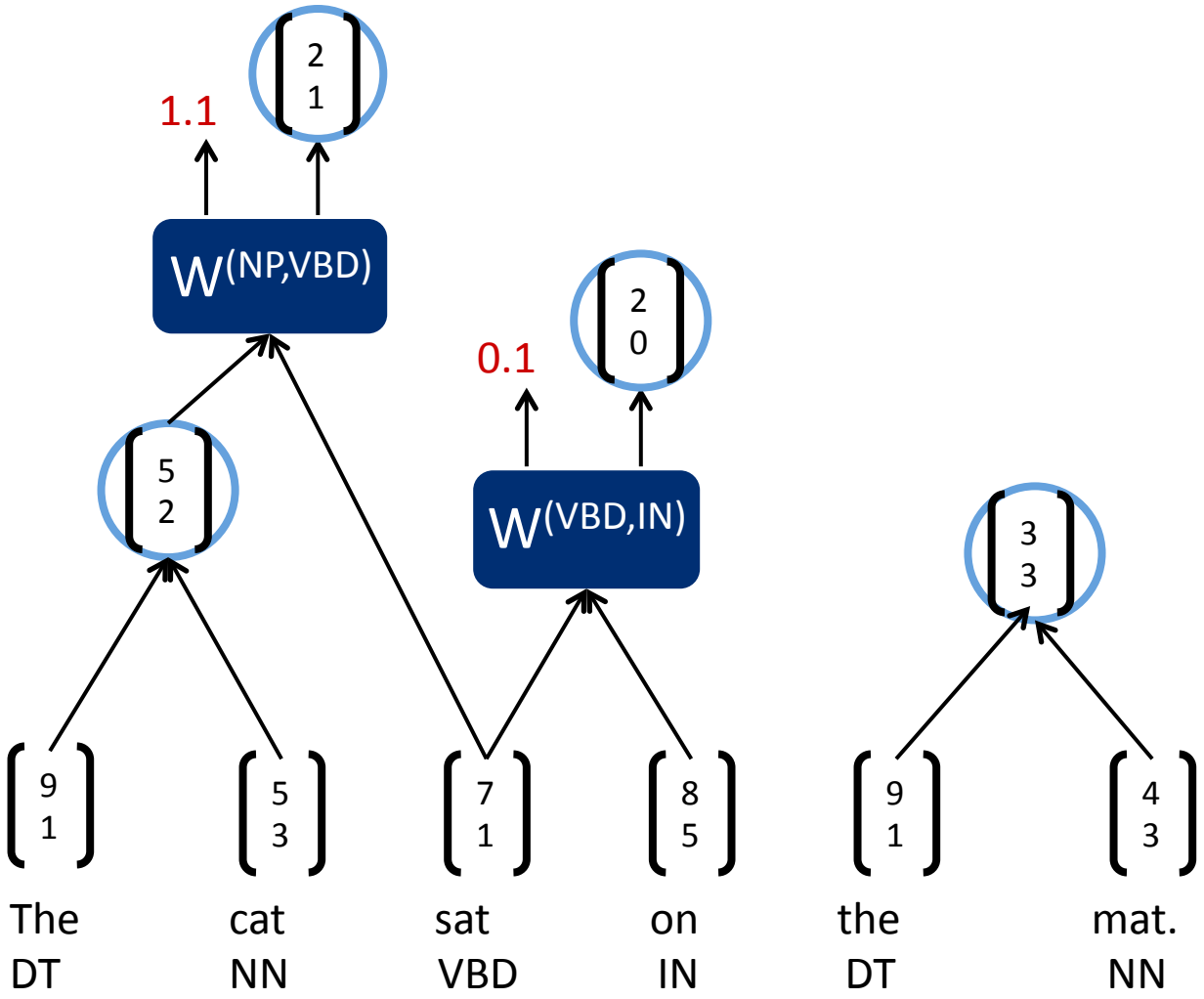
SU-RNN Procedure on Candidate Sentence



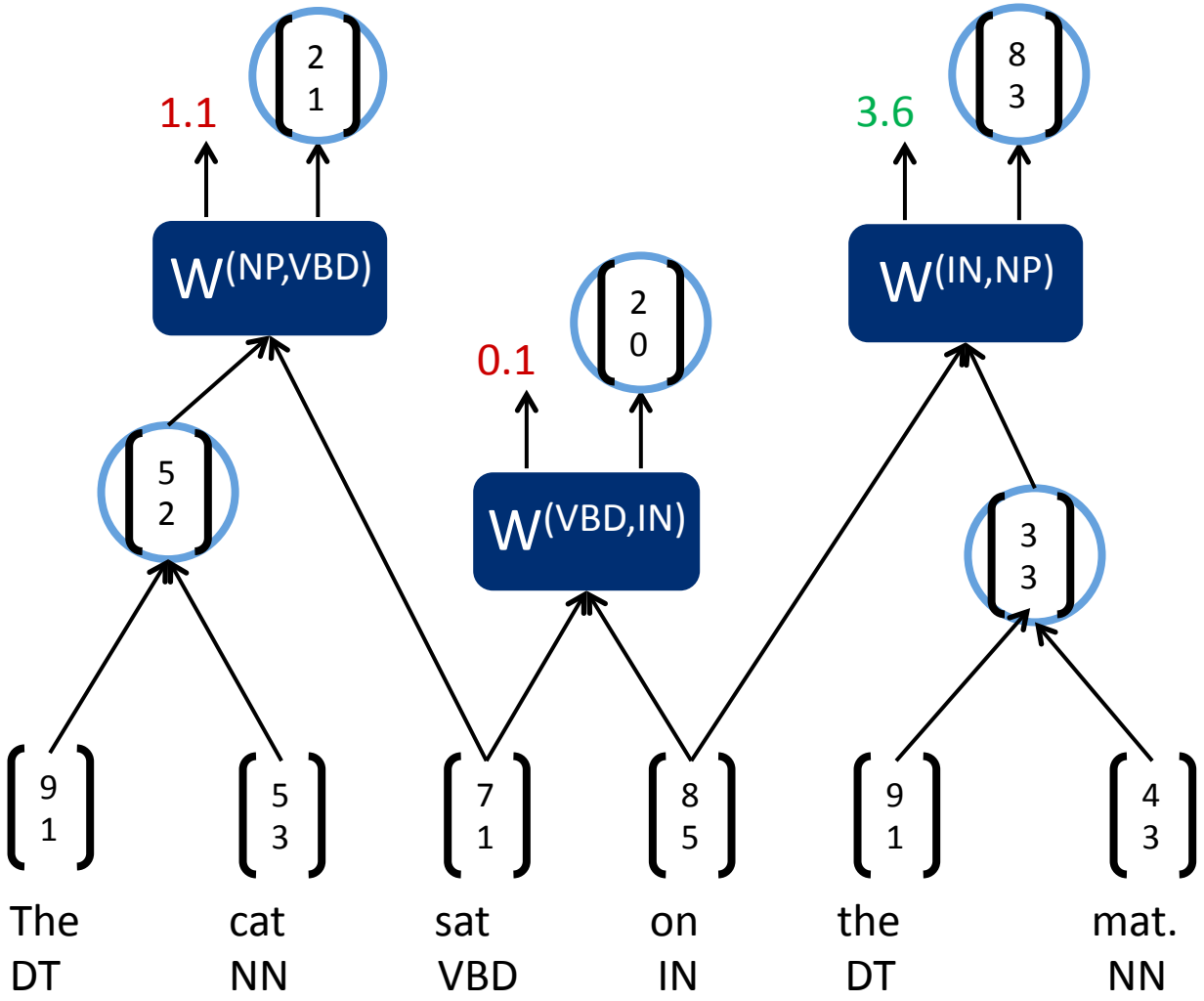
SU-RNN Procedure on Candidate Sentence



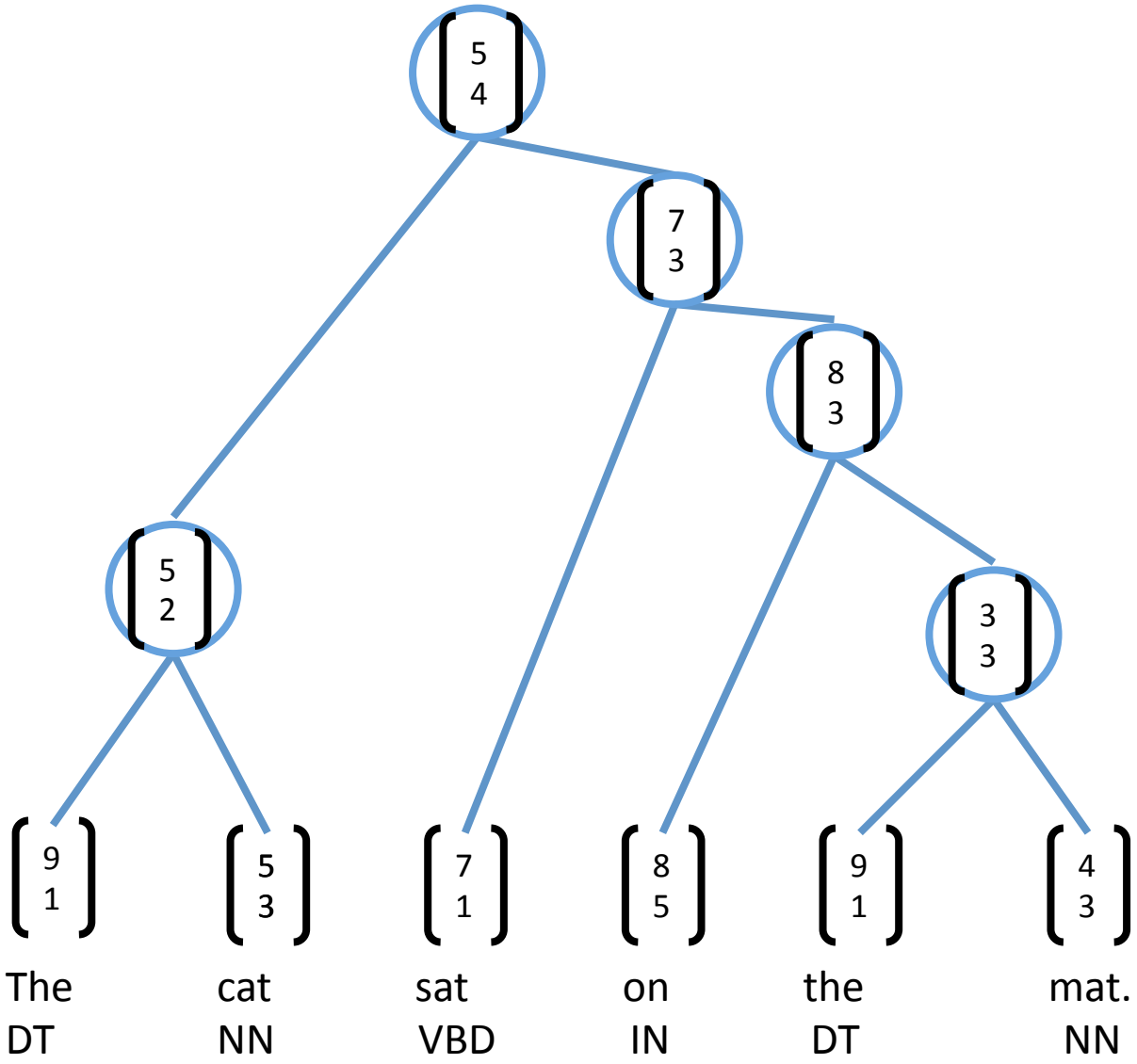
SU-RNN Procedure on Candidate Sentence



Parsing a sentence



Parsing a sentence



Details for Testing and Training CVG

- Faster to simply do *k*-best re-ranking from top 200 PCFG trees (Huang and Chiang, 2005; Charniak, 2006)

- Score of full tree *y* is sum of local scores

- Trained in a max-margin framework (Taskar et al. 2004)

$$J = \sum_i s(x_i, y_i) - \max_{y \in A(x_i)} \left(s(x_i, y) + \Delta(y, y_i) \right)$$

Experiments

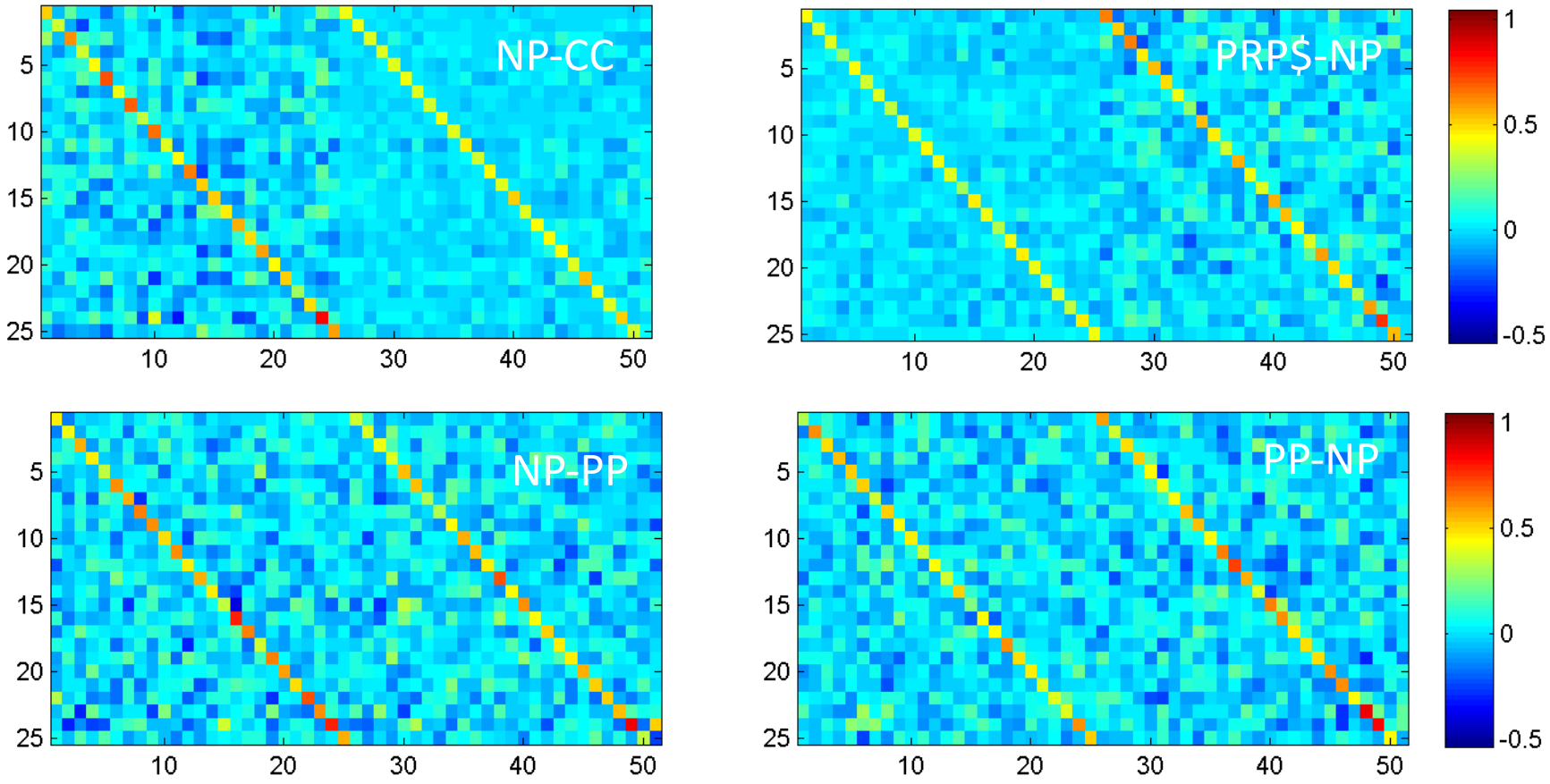
Standard *WSJ* split, labeled F_1

Parser	Test, All Sentences
Stanford PCFG, (Klein and Manning, 2003a)	85.5
Stanford Factored (Klein and Manning, 2003b)	86.6
Factored PCFGs (Hall and Klein, 2012)	89.4
Collins (Collins, 1997)	87.7
SSN (Henderson, 2004)	89.4
Berkeley Parser (Petrov and Klein, 2007)	90.1
CVG (RNN) (Socher et al., ACL 2013)	85.0
CVG (SU-RNN) (Socher et al., ACL 2013)	90.4
Charniak - Self Trained (McClosky et al. 2006)	91.0
Charniak - Self Trained-ReRanked (McClosky et al. 2006)	92.1

SU-RNN Analysis

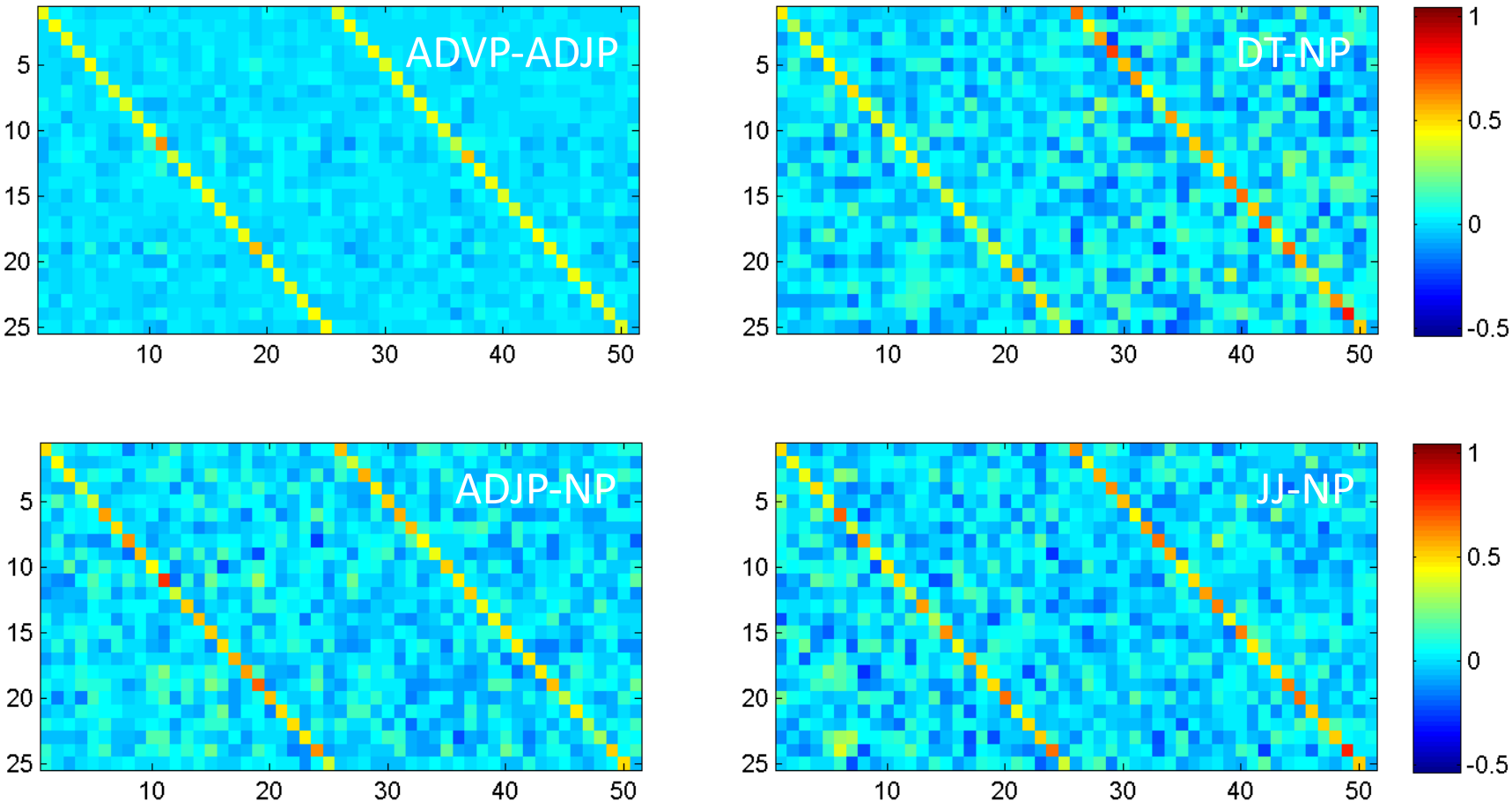
Learns soft notion of head words

Initialization: $W^{(\cdot\cdot)} = 0.5[I_{n\times n} I_{n\times n} 0_{n\times 1}] + \epsilon$



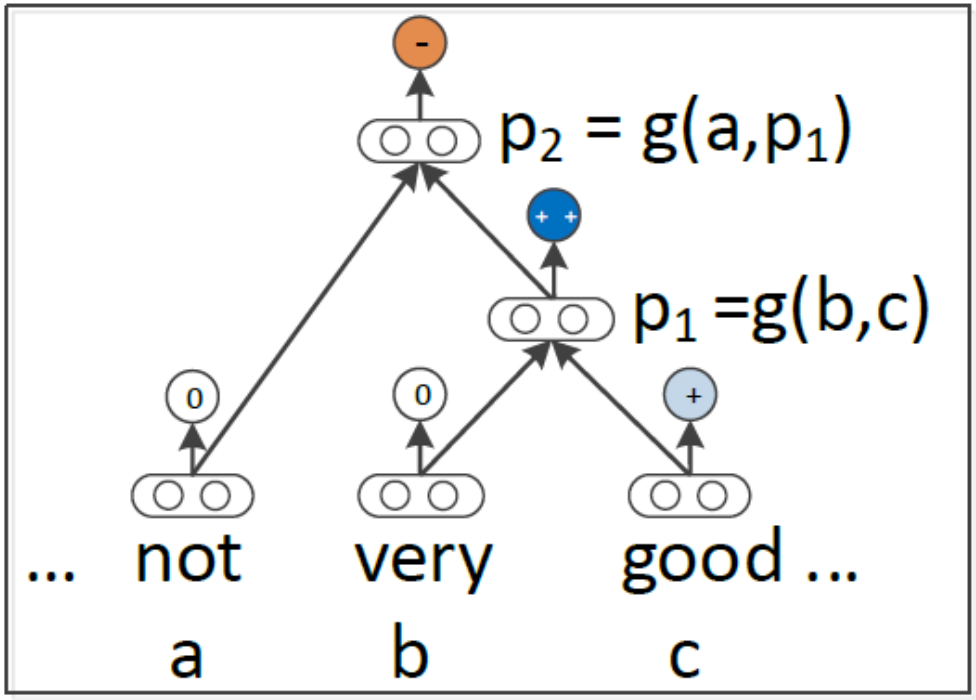
SU-RNN Analysis

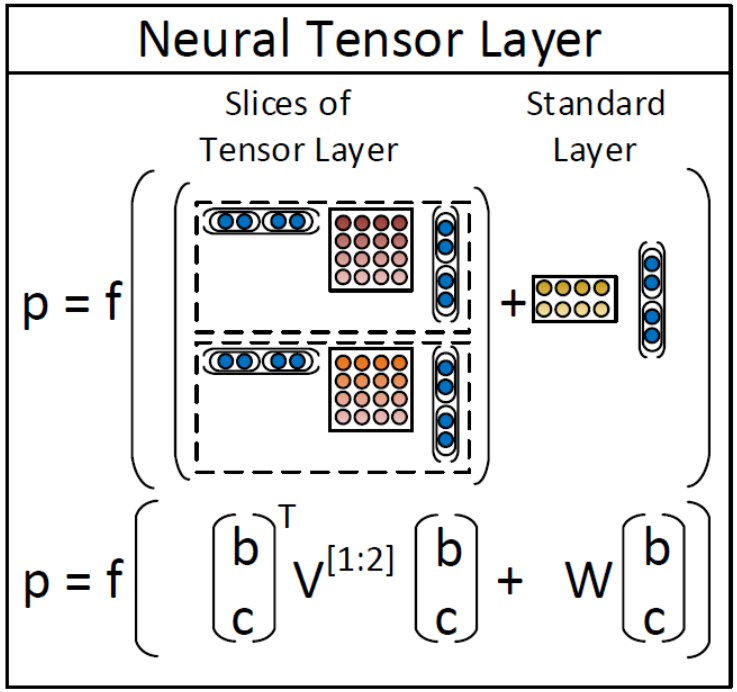
Some consistency and strength in “semantic head” modifiers



Solution 3: Recursive Neural Tensor Network

- Allows the two word or phrase vectors to interact multiplicatively





Sentiment Detection and Bag-of-Words Models

- Sentiment important for voice of the consumer, etc.
- Sentiment is that sentiment is "easy"
- Detection accuracy for longer documents ~90%

- For datas ws (Pang and Lee, **80%** for >7 years

- Harder ca of negation effects

Maybe she'll change her name to Halliburton. Just to see.

3/18/11 at 4:00 PM | 17 Comments

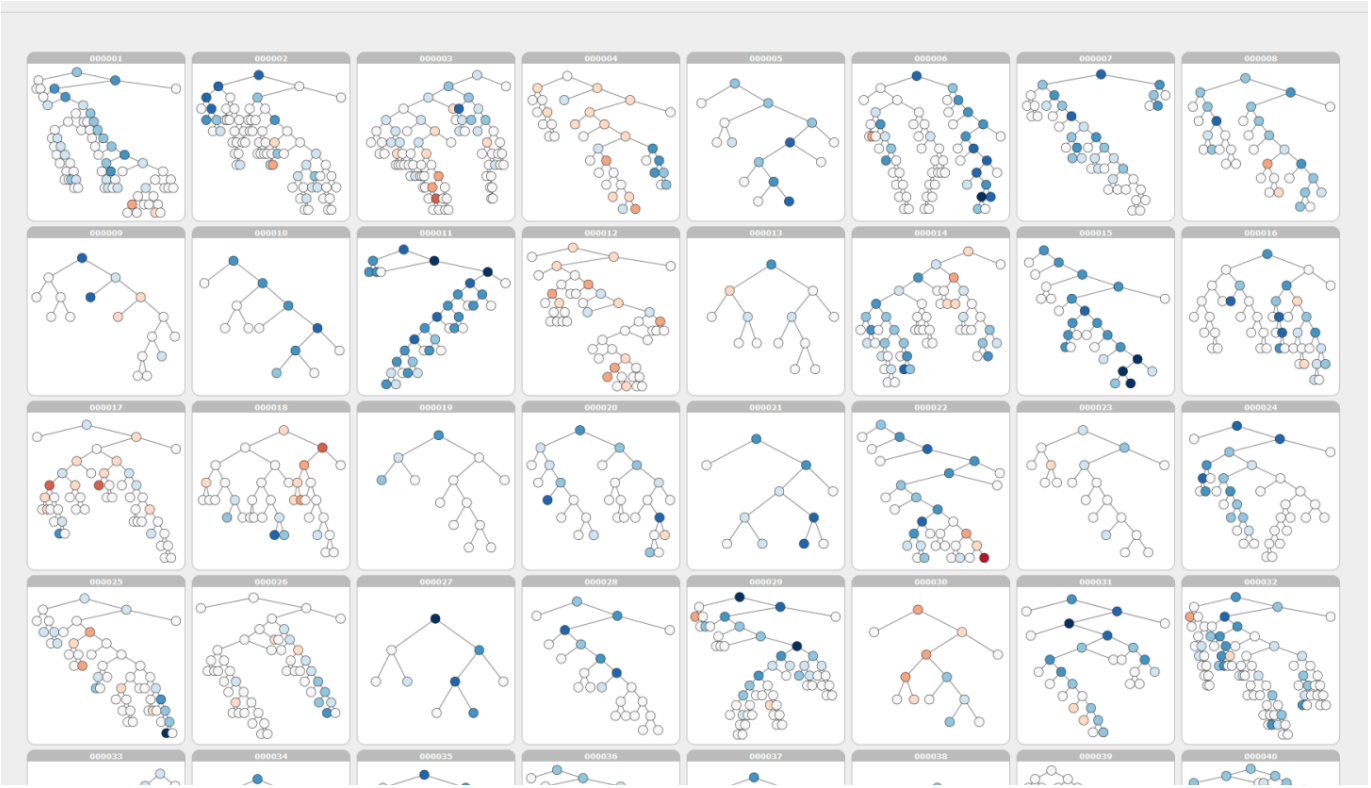
Mentions of the Name 'Anne Hathaway' May Drive Berkshire Hathaway Stock

By Patrick Huguenin

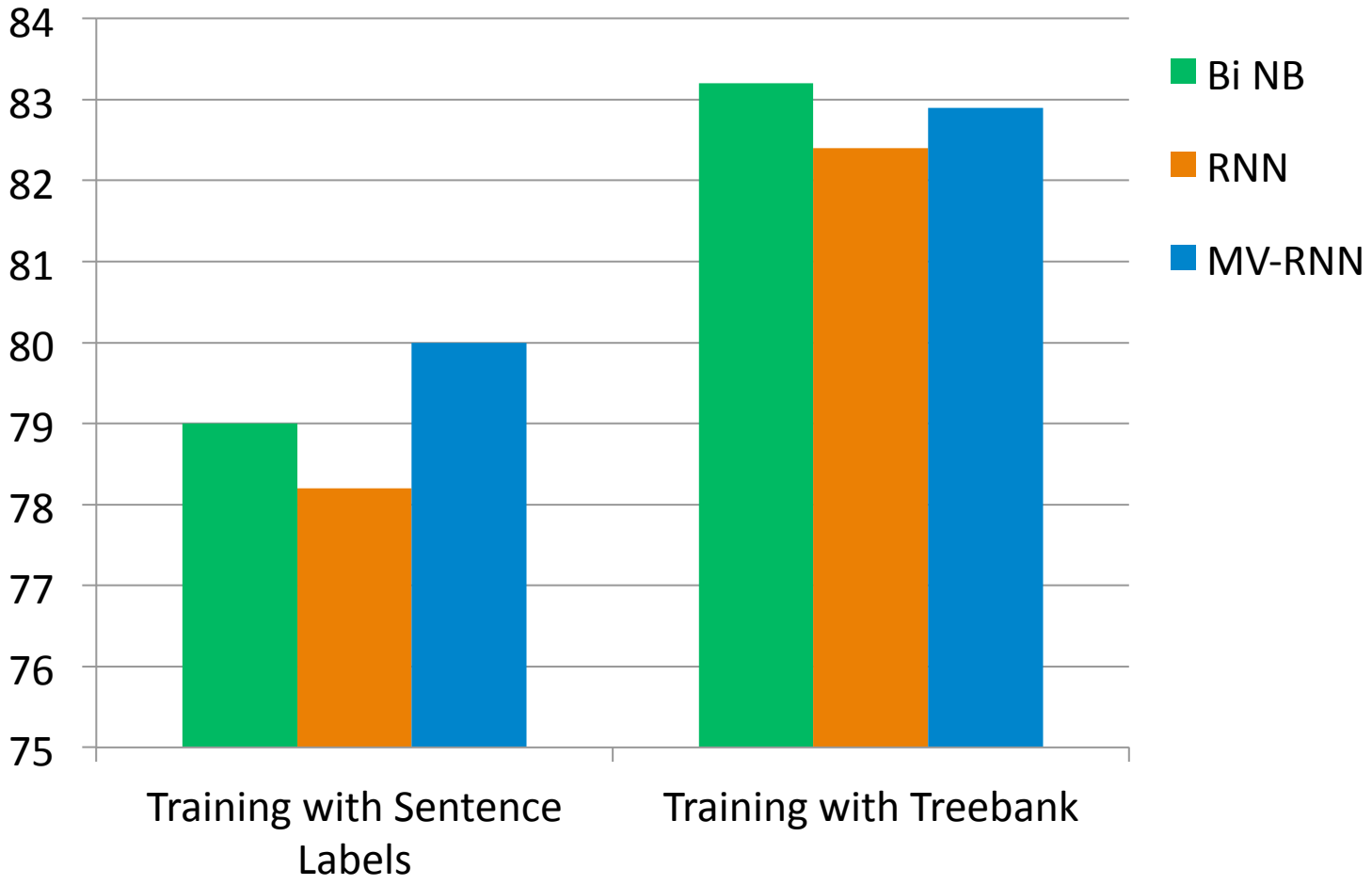
The Huffington Post recently pointed out that whenever Anne Hathaway is in the news, the stock price for Warren Buffett's Berkshire Hathaway goes up. Really. When *Bride Wars* opened, the stock rose 2.61 percent. (*Rachel Getting Married* only kicked it up 0.44 percent, but, you know, that one was so

A treebank for sentiment

- 215,154 phrases labeled in 11,855 sentences
- Can actually train and test compositions



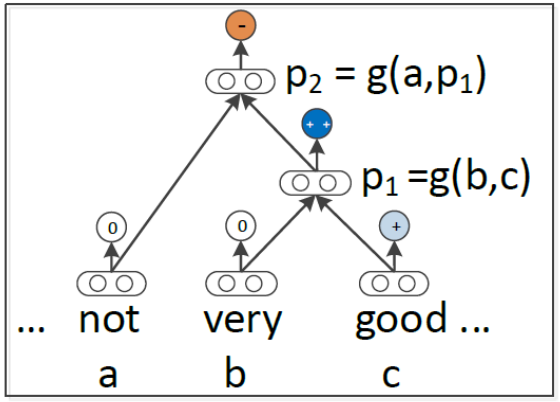
Better Dataset Helped All Models

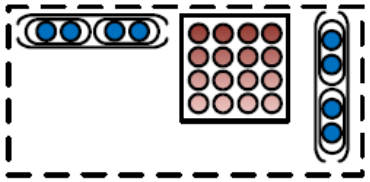


- But hard negation cases are still mostly incorrect
- We also need a more powerful model!

New Model: Recursive Neural Tensor Network

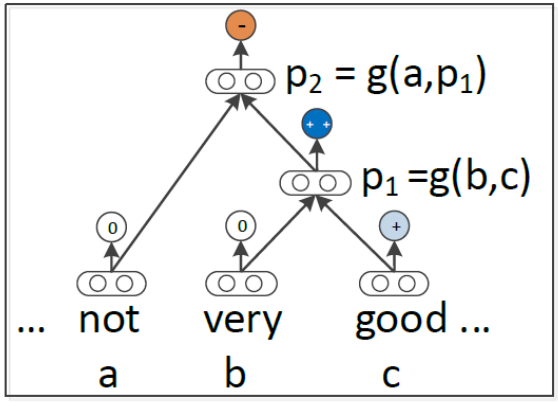
- Goal: Function that composes two vectors
- More expressive than any other RNN so far
- Idea: Allow both additive and mediated multiplicative interactions of vectors

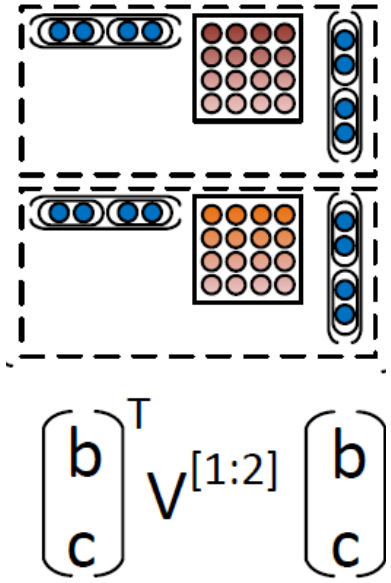




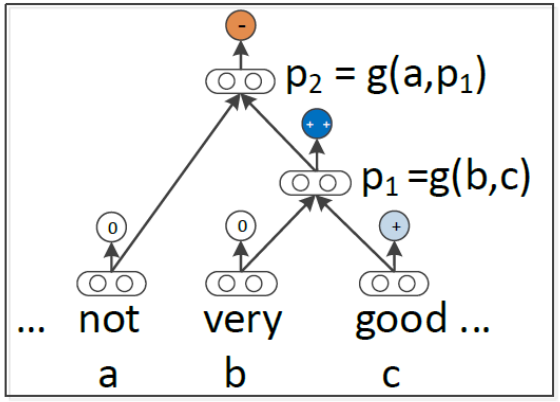
$$\begin{bmatrix} b \\ c \end{bmatrix}^T V \begin{bmatrix} b \\ c \end{bmatrix}$$

Recursive Neural Tensor Network





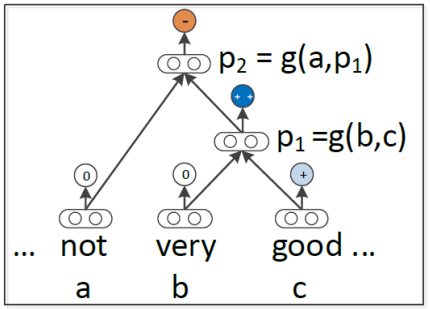
Recursive Neural Tensor Network

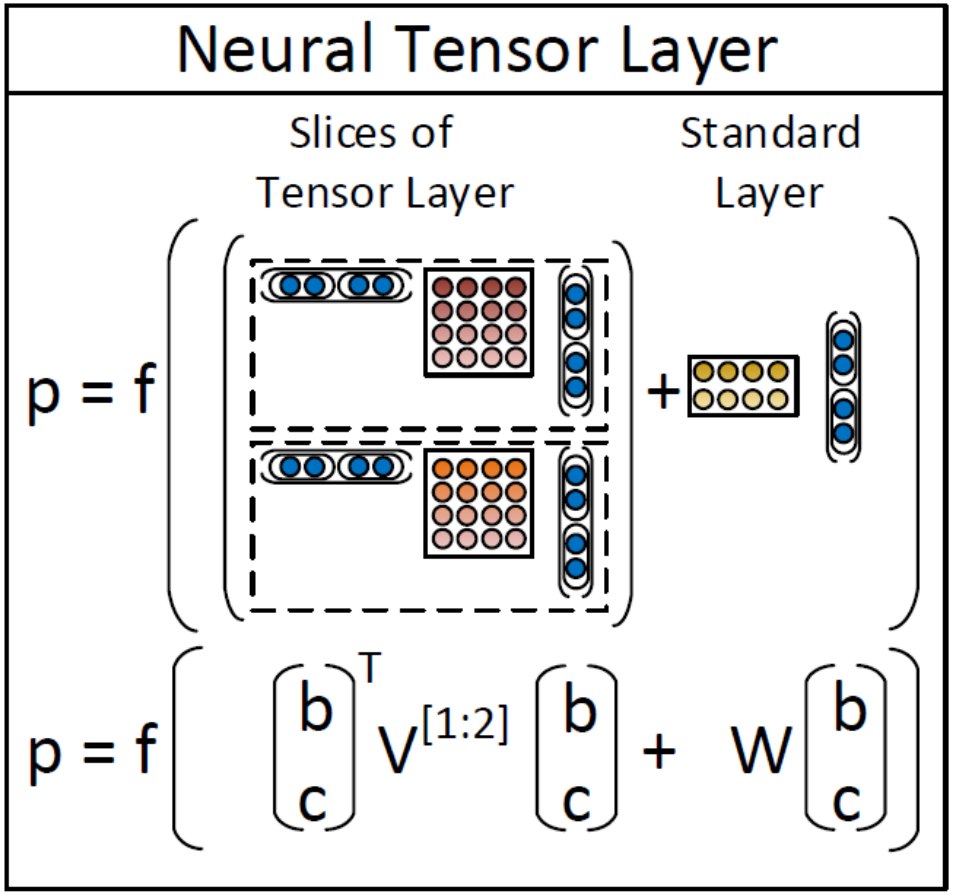


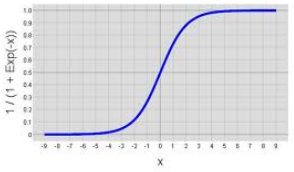
$$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}$$

Recursive Neural Tensor Network

- Use resulting vectors in tree as input to a classifier like logistic regression
- Train all weights jointly with gradient descent

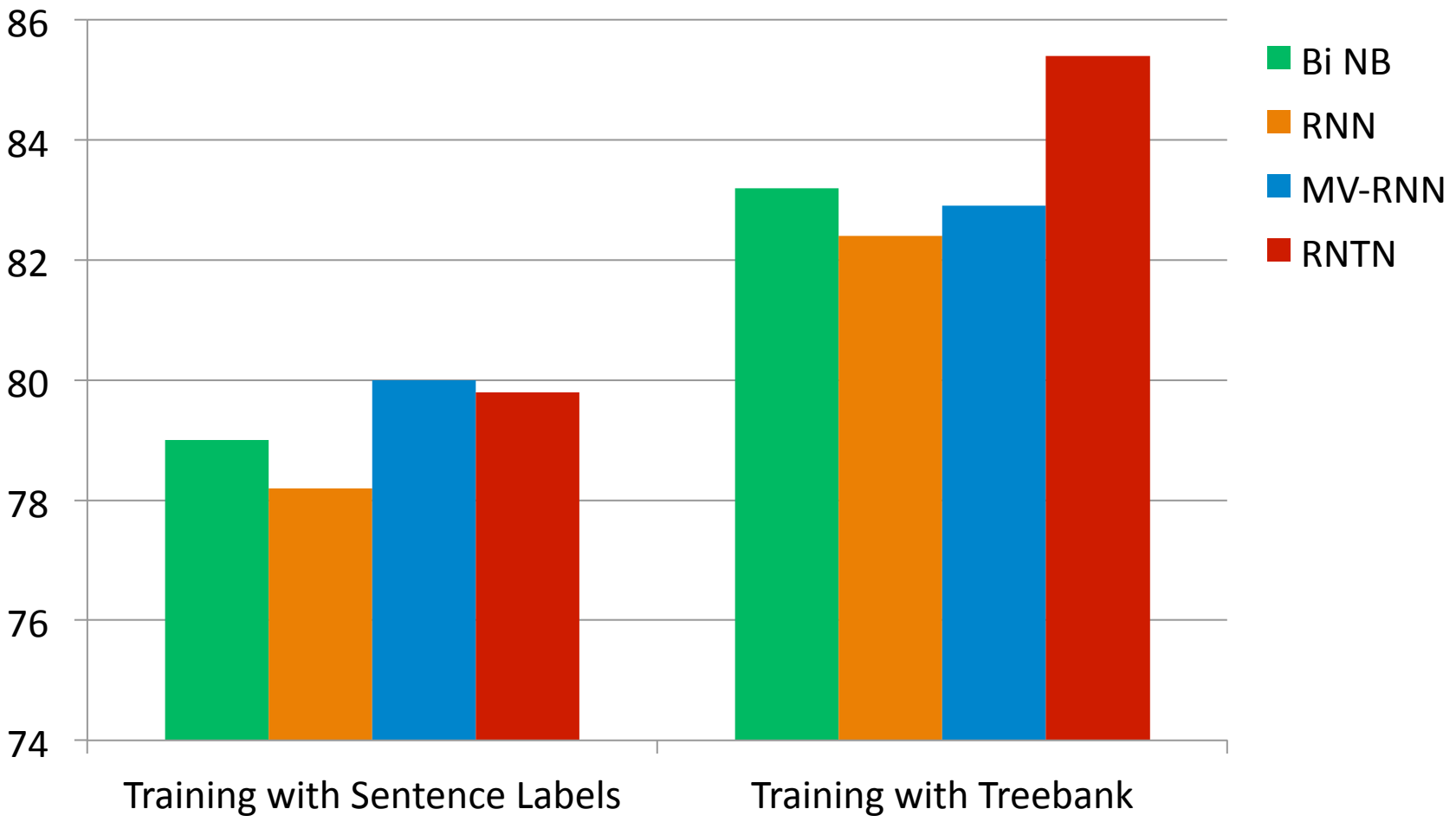






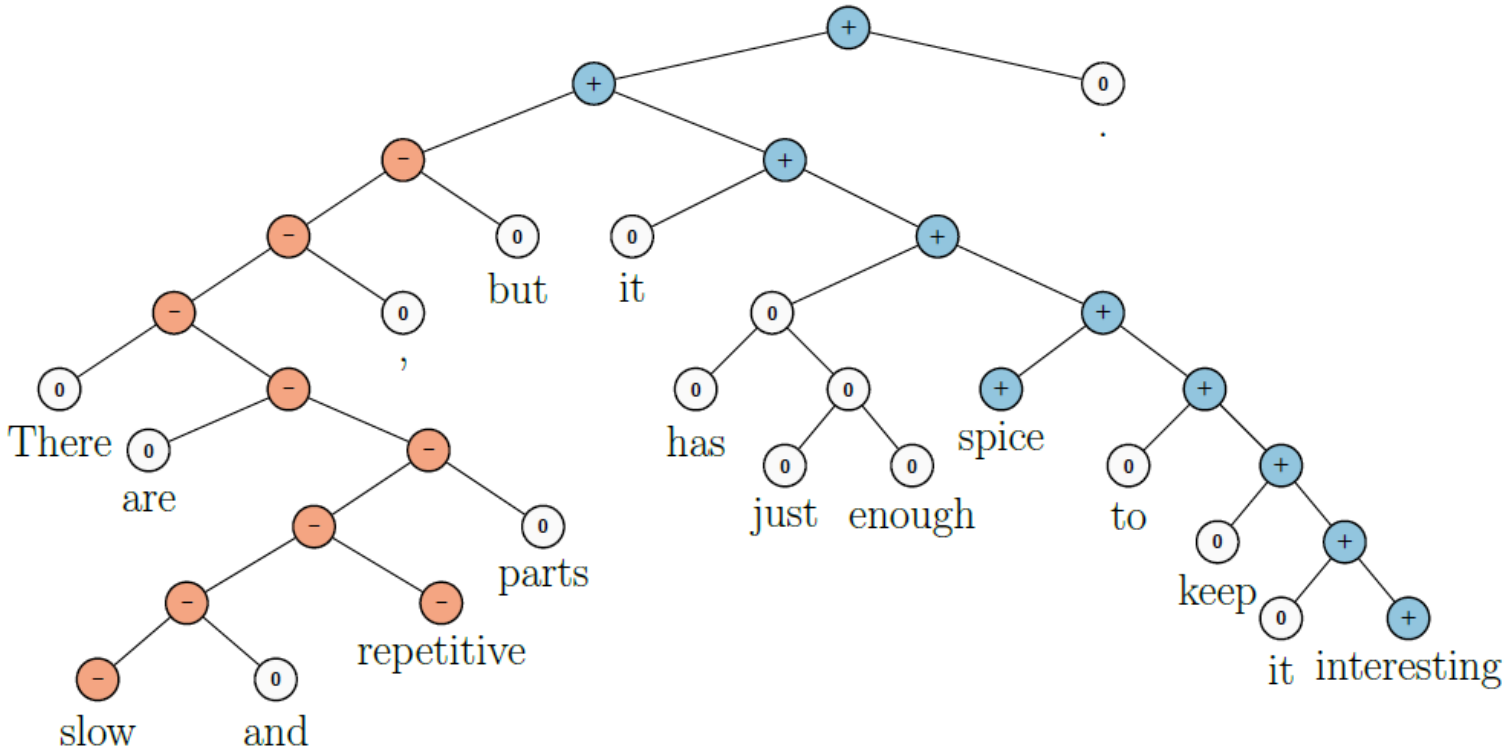
Positive/Negative Results on Treebank

Classifying Sentences: Accuracy improves to 85.4



Experimental Results on Treebank

- RNTN can capture constructions like *X but Y*
- RNTN accuracy of 72%, compared to MV-RNN (65%), biword NB (58%) and RNN (54%)

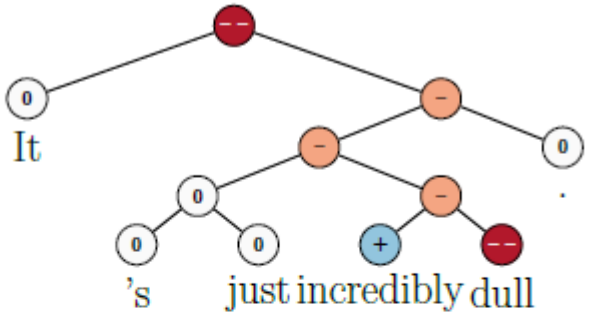


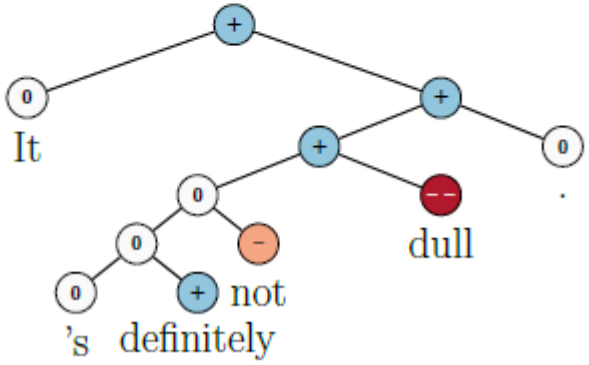
RNTN Sentiment prediction

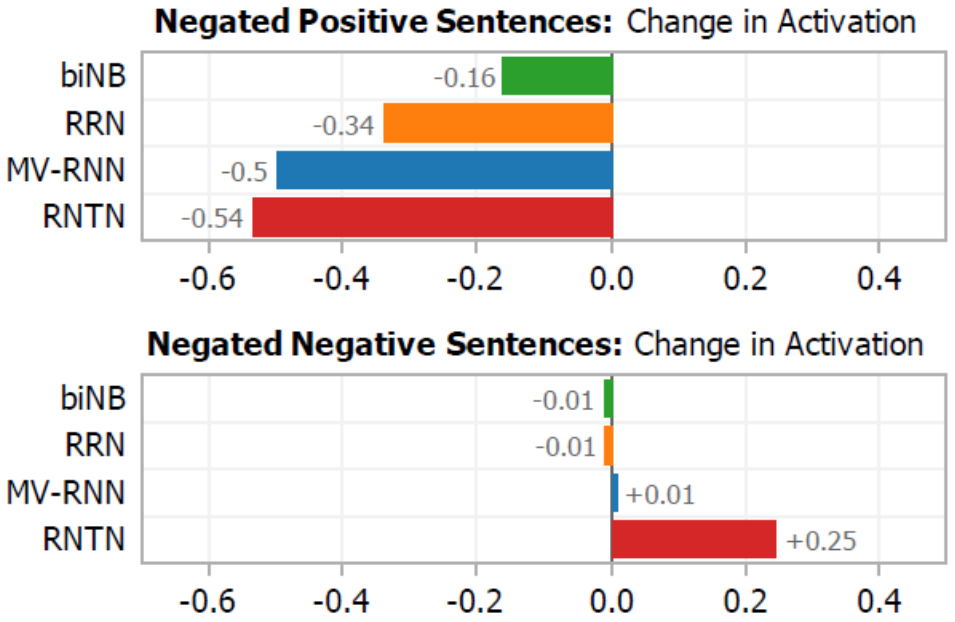
Model	Root node pos/neg
SVM	79.4
Naïve Bayes	81.8
Biword Naïve Bayes	83.1
RNN	82.4
MV-RNN	82.9
RNTN	85.4

Negation Results

When negating negatives, positive activation should increase!







Demo: http://nlp.stanford.edu:8080/sentiment/

Conclusion

Developing intelligent machines involves being able to recognize and exploit compositional structure

It also involves other things like top-down prediction, of course

Work is now underway on how to do more complex tasks than straight classification inside deep learning systems